Heterogeneous Connection: MS-SQL als DWH-Quelle, angebunden per JDBC/OpenSource-Framework ORAJDBCLINK

Günther Herzog
Metafinanz
München

Schlüsselworte:

Heterogeneous Connection, JDBC, Open Source, MS-SQL, MySQL,

Einleitung

http://orajdbclink.sourceforge.net/

In einem Projekt bestand die Aufgabe an ein ORACLE Data Warehouse neben ORACLE-Datenbanken als zusätzliche Quelle einen Microsoft SQL-Server anzubinden. Da jedoch von dem MS-SQL-Server nicht viele Daten benötigt wurden und auch daher die Priorität sehr niedrig war, sollte das Projektbudget geschont werden.
Der Datenzugriff wurde per JDBC gelöst, der von dem Open-Source-Framework ORAJDBCLINK direkt aus der ORACLE-Datenbank gesteuert wird. Dieser Vortrag stellt dieses Sourceforge Open-Source-Framework ORAJDBCLINK vor, mit dessen Hilfe aus einer ORACLE Datenbank mittels JDBC auf andere Datenbanken zugegriffen werden kann.

Features

Mit dem Framework kann man aus einer ORACLE-Datenbank alle per JDBC (JavaDataBaseConnection) erreichbaren Datenbanken erreichen. Das heißt man kann neben einer MySQL ebenso eine alte ORACLE Datenbank in der Version 7 oder früher,
die nicht mehr per Datenbanklink erreichbar ist, ansprechen. Es ist der gleichzeitige Zugriff auf mehrere JDBC-fähige Datenbanken aus einer ORACLE Datenbank möglich. Dafür sind mehrere JDBC-Treiber notwendig.

Voraussetzung

Die Voraussetzung für die Installation ist die aktivierte JavaOption JVM, deswegen ist ORACJDBCLINK auf der DB 10gXE nicht lauffähig.

Installation

Das Framework lässt sich sehr einfach installieren.
Das folgende Vorgehen bezieht sich auf eine Modifikation der Frameworksourcen um die Installation übersichtlicher zu gestalten. Diese Erweiterungen sind auf Anfrage an unten stehende Kontaktadresse zu beziehen.

1. Anlegen eines Framework-Users `JDBCLINK`, der die Funktionalitäten bereitstellt:

```
sqlplus system/manager@ora @sys_create_user_jdbclink.sql
```

Dieses Skript enthält neben dem CREATE USER die zusätzlich benötigten Grants wie das JavaUserPriv und das JavaSysPriv:

```
GRANT JAVAUSERPRIV, JAVASYSPRIV TO JDBCLINK;
```

2. Laden der benötigten JAVA Jar-Dateien der JDBC-Treiber mit loadjava. Hier im Beispiel die JTDS-Treiber für den Zugriff auf einen MS-SQL-Server 2000, auszuführen auf einer Windows-Kommandozeile:

```
loadjava -resolve -verbose -user jdbclink/jdbclink@orcl jcifs-1.3.13.jar
jtds-1.2.2.jar
```

3. Anlegen aller benötigten Javaklassen, Typen und Objekten als User `JDBCLINK`.

```
sqlplus jdbclink/jdbclink@ora @jdbclink_initoracletoany.sql
```

4. Es sind die Objektberechtigungen für die User, die das Framework benutzten dürfen zu vergeben:

```
sqlplus system/manager@ora @sys_grants.sql
```

In den originalen Frameworksourcen wird allen DB-Usern die Berechtigungen global vergeben, das Framework zu benutzen, was aber aus Sicherheitsgründen geändert wurde. Hier im Beispiel erhält in dem Skript `sys_grants.sql` nur der User des Beispielschemas `HR` die Berechtigungen ORAJDBCLINK zu benutzen.

```
grant all on jdbclink.jcursor to hr;
grant all on jdbclink.jcall to hr;
grant all on jdbclink.jdbc_dblink to hr;
```

5. Das Verschlüsselungs-Package compilieren (eine Eigenentwicklung des Projekts)

```
sqlplus jdbclink/jdbclink@ora @jdbclink_utl_decrypt.sql
```

6. Die Datenbankzugriffe auf die JDBC-Quellen in die Config-Tabelle eintragen:

```
sqlplus jdbclink/jdbclink@ora @jdbclink_insert_db_connection.sql
```

In dem Script kann man wählen, ob das Passwort im klartext oder Verschlüsselt eingetragen wird. Falls man sich für die Verschlüsselung entscheidet, ist auch die Änderung in der JavaKlasse `ConnectionConfig` vorzunehmen. Mehr dazu weiter unten.

```
-- ohne Verschluesselung des Passwortes
INSERT INTO JDBC_DBLINK (DATA_SOURCE_NAME,URL,DBUSER,DBPASSWORD,DRIVER)
VALUES
('mssql2000','jdbc:jtds:sqlserver://oradb:1433','oracle','oracle','net.sour
ceforge.jtds.jdbc.Driver');

-- mit Verschluesselung
```

```
INSERT INTO JDBC_DBLINK (DATA_SOURCE_NAME,URL,DBUSER,DBPASSWORD,DRIVER)
VALUES
('mssql2000','jdbc:jtds:sqlserver://oradb:1433','oracle',UTL_DECRYPT.ENCRYP
T('oracle'),'net.sourceforge.jtds.jdbc.Driver');
```

Funktionsweise

Für jedes SELECT wird ein Record Type angelegt der in einer separaten Funktion die die Daten per Pipelined Table Function zurückgibt. Typischerweise wird pro JDBC-Datenquelle ein Package angelegt und die Funktionen beinhalten den Namen der Quelltabelle.

```
create or replace PACKAGE HR.MSSQL AS
  TYPE EMPLOYEES_RECORD IS RECORD
  (
    "EMPLOYEE_ID"  NUMBER(6,0),
    "FIRST_NAME"   VARCHAR2(20 BYTE),
    "LAST_NAME"    VARCHAR2(25 BYTE),
    "EMAIL"        VARCHAR2(25 BYTE),
    "PHONE_NUMBER" VARCHAR2(20 BYTE),
    "HIRE_DATE" DATE,
    "JOB_ID"         VARCHAR2(10 BYTE),
    "SALARY"         NUMBER(8,2),
    "COMMISSION_PCT" NUMBER(2,2),
    "MANAGER_ID"     NUMBER(6,0),
    "DEPARTMENT_ID"  NUMBER(4,0),
    "INSERT_DATE"    DATE
  );
  TYPE EMPLOYEES_TABLE IS TABLE OF EMPLOYEES_RECORD;

  FUNCTION employees RETURN employees_table pipelined;
END MSSQL;
/

create or replace PACKAGE BODY HR.MSSQL AS
  FUNCTION EMPLOYEES RETURN EMPLOYEES_TABLE PIPELINED AS
    V_CURSORjdbclink.JCURSOR:= jdbclink.JCURSOR('select EMPLOYEE_ID
                                                      , FIRST_NAME
                                                      , LAST_NAME
                                                      , EMAIL
                                                      , PHONE_NUMBER
                                                      , HIRE_DATE
                                                      , JOB_ID, SALARY
                                                      , COMMISSION_PCT
                                                      , MANAGER_ID
                                                      , DEPARTMENT_ID
                                                 from EMPLOYEES'
                                                ,'MSSQL2000',1);
    V_RECORD              EMPLOYEES_RECORD;

  begin
  v_cursor.init; -- open connection, and prepare query
  v_cursor.open; -- execute query

  while v_cursor.dofetch = 1 loop -- fetch query results into view record
```

```
    V_RECORD.EMPLOYEE_ID      := V_CURSOR.GET_NUMBER(1);
    V_RECORD.FIRST_NAME       := V_CURSOR.GET_STRING(2);
    V_RECORD.LAST_NAME        := V_CURSOR.GET_STRING(3);
    V_RECORD.EMAIL            := V_CURSOR.GET_STRING(4);
    V_RECORD.PHONE_NUMBER     := V_CURSOR.GET_STRING(5);
    V_RECORD.HIRE_DATE        := V_CURSOR.GET_DATE(6);
    V_RECORD.JOB_ID           := V_CURSOR.GET_STRING(7);
    V_RECORD.SALARY           := V_CURSOR.GET_NUMBER(8);
    V_RECORD.COMMISSION_PCT   := V_CURSOR.GET_NUMBER(9);
    V_RECORD.MANAGER_ID       := V_CURSOR.GET_NUMBER(10);
    V_RECORD.DEPARTMENT_ID    := V_CURSOR.GET_NUMBER(11);

    pipe row (v_record); -- pipe row to the query
  end loop;

  v_cursor.close;  -- close resources

  EXCEPTION
    WHEN OTHERS THEN -- if something happens
      V_CURSOR.CLOSE;  -- close resources
       RAISE;
  END EMPLOYEES;
END MSSQL;
/
```

Abrufen der Daten

Ein Abruf der Daten ist über den Aufruf der Table-Function zu realisieren.

```
SELECT EMPLOYEE_ID
     , FIRST_NAME
     , LAST_NAME
     , EMAIL
     , PHONE_NUMBER
     , HIRE_DATE
     , JOB_ID
     , SALARY
     , COMMISSION_PCT
     , MANAGER_ID
     , DEPARTMENT_ID
     , SYSDATE
FROM TABLE(HR.MSSQL.EMPLOYEES);
```

Performance

Schnell:

```
SELECT EMPLOYEE_ID
     , FIRST_NAME
     , LAST_NAME
     , EMAIL
FROM TABLE(HR.MSSQL.EMPLOYEES)
```

Langsam:

```
SELECT EMPLOYEE_ID
     , FIRST_NAME
     , LAST_NAME
     , EMAIL
FROM TABLE(HR.MSSQL.EMPLOYEES)
where LAST_NAME = 'Baer';
```

Da die Table-Funktion ein SQL-SELECT versteckt muss bei einer Einschränkung auf ORACLE-Seite der komplette Tabelleninhalt übertragen werden und kann dann erst gefiltert werden. Eine elegantere Möglichkeit ist eine Einschränkung für dieses Attribut in einer neuen Funktion mit Parameter zur Verfügung zu stellen, was jedoch den Nachteil hat, dass man für jede Filter-Kombination eine neue Prozedur schreiben muss, was normalerweise nicht praktikabel ist:

```
FUNCTION EMPLOYEES_BY_LAST_NAME(P_LAST_NAME IN VARCHAR2) RETURN
EMPLOYEES_TABLE PIPELINED
as

v_cursor jdbclink.jcursor:= jdbclink.jcursor('select EMPLOYEE_ID, LAST_NAME
from employees where LAST_NAME= ?','MSSQL2000',2); --define the cursor
query

v_record EMPLOYEES_RECORD;

begin

v_cursor.init; -- open connection, and prepare query
V_CURSOR.BIND(1, P_LAST_NAME); -- bind code variable
v_cursor.open; -- execute query

while v_cursor.dofetch = 1 loop -- fetch query results into your view
record
    V_RECORD.EMPLOYEE_ID      := V_CURSOR.GET_NUMBER(1);
    V_RECORD.LAST_NAME        := V_CURSOR.GET_STRING(2);

pipe row (v_record); -- pipe row to the query
end loop;

v_cursor.close; -- close resources

exception
when others then -- if something happens
v_cursor.close; -- close resources
raise; -- raise the exception
```

```
end EMPLOYEES_BY_LAST_NAME;
```

Für die Übertragung von großen Datenmengen kann die Auswahl des JDBC-Treibers eine sehr wichtige Rolle Spielen. Da in dem Projekt nur sehr kleine Datenmengen übertragen wurde, war dieser Punkt irrelevant.

Füllen einer Staging-Tabelle

Für den ETL-Betrieb, ist eine Prozedur mit Parameter eher sekundär. Man kann von vorne herein das komplette SELECT inklusive Filter ausführen oder auf der Quellseite eine VIEW anlegen, die immer nur die gewünschten Daten liefert. Das Resultset wird dann in eine Staging-Tabelle geladen:

```
FUNCTION LOAD_EMPLOYEES_STG RETURN NUMBER AS
BEGIN
  DELETE STG_EMPLOYEES; --vorherige Daten löschen
  INSERT INTO STG_EMPLOYEES (EMPLOYEE_ID
                           , FIRST_NAME
                           , LAST_NAME
                           , EMAIL
                           , PHONE_NUMBER
                           , HIRE_DATE
                           , JOB_ID
                           , SALARY
                           , COMMISSION_PCT
                           , MANAGER_ID
                           , DEPARTMENT_ID
                           , INSERT_DATE)
                     SELECT EMPLOYEE_ID
                           , FIRST_NAME
                           , LAST_NAME
                           , EMAIL
                           , PHONE_NUMBER
                           , HIRE_DATE
                           , JOB_ID
                           , SALARY
                           , COMMISSION_PCT
                           , MANAGER_ID
                           , DEPARTMENT_ID
                           , SYSDATE
                    FROM TABLE(MSSQL.EMPLOYEES)
                    ;
  COMMIT;
  return 1;

  EXCEPTION
    WHEN OTHERS THEN
     ROLLBACK; --Falls es Probleme gibt, sind nach einem Rollback
               -- die vorherigen Daten wieder verfügbar.
     RETURN 0;
  END LOAD_EMPLOYEES_STG;
```

Anbinden mehrerer JDBC-Datenquellen

Das Framework unterstützt mehrere gleichzeitig nutzbare JDBC-Connections. Dazu muss lediglich in die Tabelle `JDBC_DBLINK` eine zusätzliche Datenbank eingetragen werden, die Anhand ihres `DATA_SOURCE_NAME` erkannt und benutzt werden kann. Es ist empfehlenswert, für jede angebundene Datenbank ein eigenes Package anzulegen.

Einschränkungen

ORAJDBCLINK ist nicht geeignet um Adhoc-Abfragen auf der Quelldatenbank durchzuführen, da wie oben besprochen für jedes neue SELECT eine dazugehörige Prozedur erstellt bzw. geändert werden muss, so dass man diesbezüglich nicht flexibel genug auf die Daten zugreifen kann.

Verschlüsselung des JDBC-DB-Passworts

In dem Framework ist eine Passwortverschlüsselung des JDBC-Zuganges nicht vorgesehen, so dass es hier notwendig wurde das Framework zu erweitern. Es wurde in der Konstruktormethode `ConnectionConfig` der Java-Klasse `net.sf.orajdbclink.oracletoany.ConnectionConfig` folgender Sourcecode hinzugefügt:

```
CallableStatement callableStatement = conn.prepareCall("{? = call
  UTL_DECRYPT.DECRYPT(?)}");
callableStatement.registerOutParameter(1, java.sql.Types.VARCHAR);
callableStatement.setString(2, password);
callableStatement.execute();
password= callableStatement.getString(1);
callableStatement.close();
```

In dieser Erweiterung wird das Verschlüsselte Passwort, das aus der Tabelle `JDBC_DBLINK` geholt wurde mit Hilfe des UTL_DECRYPT-Packages entschlüsselt und zurückgegeben. Dieses Verschlüsselungspackage Namens UTL_DECRYPT wurde dazu ebenso entwickelt. Als Verschlüsselung werden die Prozeduren `DES3ENCRYPT` bzw. `DES3DECRYPT` des Packages `DBMS_OBFUSCATION_TOOLKIT` in dem der 3DES-Algorithmus verwendet wird. Dieses Package steht seit der ORACLE Datenbank Version 8i zur Verfügung.
Der Key für die Verschlüsselung muss eine Mindestlänge von 128bit d.h. 16 Zeichen haben.

Die Länge einer Zeichenkette, die verschlüsselt werden soll (hier das Passwort), muss ein Vielfaches der Länge von 8 Byte besitzen, wobei das selbstentwickelte Package `UTL_DECRYPT` kürzere Zeichenketten vor der Verschlüsselung verlängert und beim entschlüsseln wieder dementsprechend kürzt. Zunächst die Verlängerung:

```
i := 8 - MOD( length('PASS'), 8 );
LV_PASSWORD2ENCRYPT := 'PASS'||RPAD(CHR(i), i, CHR(i));
```

Hier wird zuerst die Anzahl der Stellen ermittelt, die der Zeichenkette "PASS" fehlen, um auf eine Länge zu gelangen, die einem Vielfachen der Länge von 8 Byte entspricht. Anschließend wird die ermittelte Anzahl an aufzufüllenden Stellen als ASCII-Code interpretiert und der CHR Funktion übergeben.

Die CHR Funktion liefert zu einem ASCII-Code das entsprechende Zeichen aus der ASCII-Tabelle zurück. Danach wird die "PASS"- Zeichenkette so oft um das Zeichen aus der ASCII-Tabelle erweitert, bis die Gesamtlänge der Zeichenkette ein Vielfaches der Länge von 8 Byte besitzt.

Für die Entschlüsselung der Zeichenkette gilt das analoge Verfahren. Dabei muss der Klartext um die erweiterten Zeichen aus der ASCII-Tabelle gekürzt werden:.

```
V_LAENGE   := LENGTH(V_KLARTEXT);
V_KLARTEXT := RPAD(V_KLARTEXT,V_LAENGE-ASCII(SUBSTR(V_KLARTEXT, V_LAENGE)));
```

Bei dieser Vorgehensweise wird die Zeichenkette immer nur um Steuerzeichen wie z. B. ^E oder ^G erweitert, die in Zeichenketten nur selten vorkommen.

Ablage des Verschlüsselung-Keys
Der Verschlüsselungs-Key wurde in dem Projekt im Verschlüsselungspackage selbst abgelegt.
Somit ist das Passwort in der Tabelle sicher verschlüsselt, und der zugehörige Key kann von anderen Usern nicht im UTL_DECRYPT-Package gelesen werden, weil hier nur der Owner JDBCLINK Ausführberechtigung hat.
Ein kleinerNachteil besteht aber darin, dass für alle Verschlüsselungen immer derselbe Verschlüsselungskey verwendet wird.

Fazit

ORAJDBCLINK ist ein nettes kleines Framework, das bei kleineren Datenaufkommen sehr schnell und unproblematisch neue Datenquellen an ein Datawarehouse anbinden kann ohne in große kommerzielle Produkte investieren zu müssen.

http://orajdbclink.sourceforge.net/

Kontaktadresse:

Günther Herzog
Metafinanz
Leopoldstrasse, 146
D-80804 München

Telefon: +49 (0) 89-360 531 5138
Fax: +49 (0) 89-360 531 15
E-Mail guenther.herzog@metafinanz.de
Internet: www.metafinanz.de